Scalable Regression Tree Learning in Data Streams

Konstantin Kutzkov, Mathias Niepert, Mohamed Ahmed

***Abstract*—We present new algorithms for scalable regression tree learning in data streams. Leveraging state-of-the-art data summarization techniques, the new algorithms work by learning a regression tree from compact sketches of the original data. The algorithms require a single pass over the data and can learn a regression tree from massive speed data streams in real time. We obtain precise theoretical bounds on the complexity and accuracy of the algorithms for two widely used loss functions, mean squared error and least absolute deviation. The approach is particularly suited for learning trees of small depth and therefore can be applied to ensemble tree learning methods such as random forests and boosted trees. Experimental evaluation on real and synthetic data confirms the theoretical findings and demonstrates the practical importance of the proposed methods.**

I. Introduction

The intuitive idea to partition the data based on different features and learn a model in each partition applies to numerous real-life problems. Unsurprisingly, classification and regression trees are among the most widely studied and applied non-parametric machine learning methods. In the era of Big data, many classic machine learning algorithms need to address the critical issue of scalability. Traditional approaches include subsampling, dimensionality reduction, feature selection and data summarization. In this paper we present new summarization algorithms that use compact *sketches* preserving the key properties of the original data. We show that we can learn a regression tree from the sketches that will approximate a regression tree learned from the entire data set. The method is particularly well suited for handling massive high speed data streams in real time.

Previous work. Domingos and Hulten [1] presented the first algorithm for decision and regression tree learning in data streams. The algorithm works by incrementally building a tree. Incoming examples are navigated to the leafs until there are enough samples to decide on a splitting. It is argued that the stream is generated from unknown but fixed distribution and examples represent samples independently drawn from it. Under this assumption, theoretical guarantees are obtained for the number of required examples to make an approximately correct splitting decision. An immediate problem is that the assumptions are often not realistic, therefore the sampling approach was extended to consider concept drift and learn regression trees allowing to revert a bad splitting [2].

Ben-Haim and Tom-Tov [3] present a decision tree learning algorithm that works on histograms of the data. The algorithm can be seen as a milti-pass streaming algorithm that incrementally builds a decision tree. Even if no precise bounds on the required number of histogram bins are shown, the algorithm yields excellent results and adapts to different feature distributions. Kpotufe and Orabona [4] present an algorithm for learning a tree based regressor for a function f which satisfies a Lipschitz condition. The main contribution is an online algorithm for high dimensional data with low intrinsic dimensionality such that examples that are close to each other in the low-dimensional space are assigned to the same leaf in the tree. The algorithm does not achieve space savings in terms of the size of the original data. Building upon AMS sketching [5], Yu et al. [6] present a sketching algorithm that learns the average weight of the target variable of examples described by pairs of discrete valued features. Essentially, this means that if the original data contains n features and examples are described by k discrete feature values, then we can learn a model that is described by pairs of $O(n^{k/2})$ discrete features.

Our Contribution. There are two major drawbacks of sampling based algorithms when applied to regression tree learning. First, it is not immediately clear how to address arbitrary distributions of the weight of the target variable. Second, there might exist highly predictive but less frequent feature values combinations that are likely to be underrepresented in the sample.

A recent line of research is to design scalable machine learning algorithms using data summarization technques. Clustering massive data using coresets [7], SVD decomposition based on frequent items mining [8], non-linear kernel approximation by sketching of outer vector products [9] and efficient recommendation in collaborative filtering systems using locality sensitive hashing techniques [10] showcase the potential which sketching methods can have on scaling machine learning algorithms. We contribute to this research direction and design novel regression tree learning algorithms using advanced data summarization techniques. Our approach is drastically different from sampling algorithms as we consider all examples in order to summarize the data. We design learning algorithms for two widely used loss functions, mean squared error and least absolute deviation, and obtain precise theoretical bounds on the space complexity of the algorithms. We originally assume discrete features but we also present discretization methods that allow us to handle real valued features. In order to achieve efficient running time, we present new algorithms for the evaluation of min-wise independent hash functions over a sequence of consecutive integers, a problem that can be of independent interest. An experimental evaluation on real and synthetic data confirms the theoretical findings.

Organization of the paper. Necessary definitions are presented in Section II. An overview of the main ideas is in Section III. In Section IV we present the general structure of the new algorithms and the used sketching techniques. The

summarization algorithm for learning trees based on the mean squared error as a loss function is presented and analyzed in Section V, and the algorithm based on least absolute deviation is in Section VI. In Section VII we consider the time complexity and design a new algorithm for the efficient evaluation of a hash function on a set of consecutive integers. The algorithms originally assumes discrete features and in Section VIII we present methods that generalize to real features. We present experimental results in Section IX. The paper is concluded in Section X where we discuss future research directions.

II. Preliminaries

Problem definition. Let $\mathcal{S} = e_1, e_2, \ldots$ be a continuous stream of training examples. For each example it holds $e_i = (\mathbf{x}_i, w(y_i))$, where $\mathbf{x}_i = (f_1 = x_1, \ldots, f_k = x_k)$ is a k-dimensional vector of feature assignments and $w(y_i) \in \mathbb{N}$ the weight of the dependent or *target variable* y_i of example e_i. We assume that each feature f can be assigned values x_j drawn from a finite domain X_f. We consider sets of feature assignments in conjunctive normal form (CNF), i.e., a conjunction of disjunctions, where disjunctions represent different possible assignments to a feature. A given CNF $(x_1^1 \vee \ldots \vee x_1^{k_1}) \wedge \ldots \wedge (x_t^1 \vee \ldots \vee x_t^{k_t})$ of feature assignments is called a *profile*. (In the CNF representation $x_j^1 \vee \ldots \vee x_j^{k_j}$ is a shorthand for $f_j = x_j^1 \vee \ldots \vee f_j = x_j^{k_j}$ and $x_j^\ell \in X_{f_j}$ is one of the $|X_{f_j}|$ possible assignments to f_j.) We say that a given profile P *complies* with a training example $e_i = (\mathbf{x}_i, w(y_i))$ iff for each each disjunction $x_j^1 \vee \ldots x_j^{k_j}$ in P there exists a feature assignment $f_j = x_j^\ell$ in $\mathbf{x}_i$, $1 \leq \ell \leq k_j$. An example e_i complying with P is denoted as $e_i \rhd P$. In a regression tree internal nodes and leaves correspond to different profiles. Each example e_i is assigned to the leaf with corresponding profile P such that $e_i \rhd P$.

Example. Assume the data consists of flight records. A record is described by the features origin, destination, week day and weather at departure. The target variable for a record is the delay from the scheduled arrival time. Consider two examples $e_1 = [(\text{Frankfurt, New York, Sunday, cloudy}), 67]$, denoting that the flight from Frankfurt to New York on a cloudy Sunday had a delay of 67 minutes, and $e_2 =$[(Munich, Barcelona, Monday, sunny), 10]. The example e_1 complies with the profile
[origin = (Frankfurt $\vee$ Munich $\vee$ Berlin) $\wedge$ destination=(New York $\vee$ Boston $\vee$ Washington DC) $\wedge$ day=(Saturday $\vee$ Sunday) $\wedge$ weather=(cloudy $\vee$ raining)]
but the example e_2 does not. Both examples comply with the profile [origin = (Frankfurt $\vee$ Berlin $\vee$ Munich)]. The goal is to learn a model that predicts the target variable *delay* for different profiles that best describe the data distribution.

Notation. The examples in $\mathcal{S}$ complying with a profile P are denoted as $\mathcal{S}_P$. Abusing notation, when clear from the context we write P for both the profile P and the examples in $\mathcal{S}_P$. $\|P\|_\ell = \sum_{e_i \rhd P} w(y_i)^\ell$, i.e., $\|P\|_\ell$ is the ℓ-th power of the ℓ-norm of the vector of target weights in $\mathcal{S}_P$.

We say that an algorithm returns an (ε, δ)-*approximation* of some quantity q if it returns a value $\tilde{q}$ such that $(1-\varepsilon)q \leq \tilde{q} \leq (1+\varepsilon)q$ with probability at least $1-\delta$ for $0 < \varepsilon, \delta < 1$.

Regression trees. We assume binary regression trees. A split in a given node divides the data in the node in two disjoint subsets. A node in a regression tree contains the examples complying with a given profile, the root being the empty profile complying with all examples. The profiles of the two children nodes extend the profile of a parent node with a disjunction of feature assignments. For a split like origin = 'Frankfurt' vs. origin != 'Frankfurt' the second branch contains a disjunction of all origin airports in the dataset different from 'Frankfurt'. Each example in the dataset can be assigned to a unique leaf in the tree. Let $\mathcal{P}$ be the set of possible profiles. At each leaf we maintain a prediction function $f : \mathcal{P} \to \mathbb{R}$ for the corresponding profile. We define a loss function $L : \mathcal{P} \to \mathbb{R}$. For example, a widely used choice for f is the mean $\mu(P)$ of target weights of examples complying with a given profile P, and for the loss function L – the mean squared error $MSE(P) = \frac{1}{\|P\|_0} \sum_{e_i \rhd P} (w(y_i) - \mu(P))^2$. We split a leaf of the tree by extending the corresponding profile with a disjunction of features that yield the maximum reduction in error for the given loss function.

III. Overview

Motivation. Our original motivation was to improve the algorithm presented by Yu et al. [6]. Assume a high-speed network stream of cell phone calls packets. Each packet is described by a set of discrete valued features, e.g. device model, operating system, origin, destination. The measure of interest, or target, is the round-trip time (RTT) of packets. The goal is real time anomaly detection of the RTT distribution for different feature profiles. For example, it is normal that certain origin-destination pairs are associated with much higher round-trip time. This is achieved by measuring the empirical variance of RTT, $\|P\|_2/\|P\|_0$, of a profile P. A sudden change in these values indicates an anomaly. The main challenge is to detect such anomalies in real time. The number of occurring profiles can easily become huge and a standard solution like storing all profiles in a hash table are not feasible. As a main contribution, Yu et al. [6] present a solution that "square roots" the number of profiles that need to be explicitly stored. Using AMS sketching [5], the authors show that sketches of two profiles P_1 and P_2 with disjoint feature sets can be combined into a sketch of the merged profile $P_1 \wedge P_2$.

Unfortunately, AMS sketches are limited to combining the sketches of pairs of profiles. Instead, we realized that a modification of min-wise independent sketches [13] is more suitable for the problem. Given the sketches of several profiles, we show that we can obtain a sketch for arbitrary conjunctions and disjunctions of profiles. Moreover, by estimating the $\|P\|_\ell$ values for $\ell = \{0, 1, 2\}$, we can estimate the distribution of different functions of the target weights. This allows us to generalize the problem to regression tree learning, a problem covering a wider set of applications.

Main ideas. Before presenting formal results, let us give a high level explanation of the new algorithms and the achieved approximation guarantees. Regression trees partition the data into subsets depending on feature values such that leafs represent a subset of training examples that comply with a given profile. In each leaf, we fit a model that best describes the data. The most prevalent version of regression trees uses a single numerical value for the prediction of the target weight of examples falling in a leaf. Usually, the prediction is the mean or the median value of the weights $w(y_i)$ of examples complying with the leaf profile. In order to decide on an optimal partition, we generate candidate splits and find a prediction that is as accurate as possible with respect to a given loss function. Clearly, the naïve algorithm is very inefficient for massive datasets as we need to access all training examples for each candidate split.

A simple solution to learn a regression tree for MSE loss of depth d in data streams could work as follows. We store in a hashtable all profiles with up to d disjunctions that are observed to occur in the stream. For each profile P in the hash table we update the values $\|P\|_\ell$ for $\ell \in \{0, 1, 2\}$. For example, for an incoming example such as e_i =[(Munich, Barcelona, Monday, sunny), 10] and $d = 4$, we will add to $\|P\|_\ell$ the values 1, 10 and 100, respectively, for each of the $2^4 - 1$ nonempty profiles occurring in e_i. It is a simple observation that the mean $\mu(P)$ and the error $MSE(P)$ can be computed from the values $\|P\|_\ell$. Therefore, we have the required information to learn a regression tree. However, explicitly storing all occurring profiles is infeasible for a larger number of possible feature values.

As a main contribution, we design algorithms that estimate the prediction and loss function from compact summaries of the data. More specifically, we keep a summary for each feature value, e.g., destination='London' or weather='sunny'.

Applications.

- The algorithm applies to learning boosted regression trees in a streaming setting as follows. For a prefix of the stream we learn a regression tree. Once the tree has been learnt we apply it to the next chunk of the stream and update the target weights $w(y_i)$ of the new examples e_i using a nonnegative loss function such as mean squared error and least absolute deviation. The algorithm also applies to random forests where we learn trees of small depth using feature bagging, i.e., each tree is learnt from a random subset of the features.
- The primary purpose of the presented algorithm is to learn a regression tree in a single pass over high-speed streaming data. The algorithm is most suitable for learning trees of small depth. However, if we can afford several passes over the data, then we can iteratively learn a tree of arbitrary depth. Assume in the i-th pass we have learned a tree T_i, T_0 being the empty tree. In the $(i+1)$-th pass we navigate examples to the corresponding leafs and in each leaf we learn a new tree of small depth.

BUILDREGRESSIONTREE

Input: sketches $sk[x_1], \ldots, sk[x_n]$, max depth d, error threshold δ

1: $T = \emptyset$
2: $P = \emptyset$
3: $optimal = False$
4: **while** $T.depth < d$ and not optimal **do**
5: $optimal = True$
6: **for** all leafs $T[P] \in T$ **do**
7: select a feature f
8: $err^* = error_P$
9: $V_l^* = V_r^* = \emptyset$
10: **for** $(V_l \cup V_r) \in$ PARTITION(X_f) **do**
11: $sk_l = \cup_{x_i \in V_l} sk[x_i]$
12: $sk_r = \cup_{x_i \in V_r} sk[x_i]$
13: $pred\_est =$ ESTIMATEPREDICTION(sk_l, sk_r)
14: $est =$ ESTIMATEERROR$(sk_l, sk_r, pred_est)$
15: **if** $err^* < est$ **then**
16: $err^* = est$
17: $V_l^* = V_l$, $V_r^* = V_r$
18: **if** $error_P - err^* \geq \delta$ **then**
19: $optimal = False$
20: $P_l = P \wedge V_l^*$
21: $P_r = P \wedge V_r^*$
22: Split as $T[P_l], T[P_r]$

Fig. 1. Regression tree learnt from sketches.

IV. PROPOSED SOLUTION

A. Regression tree learnt from sketches

Pseudocode for the proposed algorithm is given in Figure 1. We assume the data stream has already been processed and there exists a sketch $sk(x_i)$ for each feature value x_i, e.g., sk(origin='London'). We run a standard regression tree algorithm but instead of using the original data we estimate the quality of a split from the sketches. The tree is denoted as T and an internal node or a leaf representing a profile P – as $T[P]$. The variable *optimal* denotes whether the prediction in a given leaf is good enough. We select a feature f according to some criteria and then generate candidate splits from the feature values sketches $sk(x_i) \in X_f$. In line 10 we assume an algorithm PARTITION that generates a candidate partition of the feature values according to some criteria, e.g. [11]. A new branch extends an existing profile with a disjunction of feature assignments, therefore we merge the required sketches $sk(x_i)$. (The algorithm for merging the sketches is presented in the next subsection.) We estimate the prediction and corresponding loss function from the sketches in lines 13 and 14, respectively. We continue splitting until either the maximal depth d is reached, or the estimated error reduction is less than δ, for user defined parameters d and δ.

B. Data summarization

The main building block in our algorithm will be the estimation of $\|P\|_\ell$ for different profiles P. We will later

show the mean and median for a profile P, as well as the MSE(P) and LAD(P) loss functions can be computed in terms of the values $\|P\|_\ell$. In the following we assume that the target weights $w(y_i)$ are integer numbers. We consider each example with weight w as the continuous arrival of w^ℓ consecutively numbered unweighted examples. With each feature value we associate a set of integers and summarize these sets. Consider the three examples [(Frankfurt, New York, Sunday, cloudy), 67], [(London, New York, Monday, sunny), 27], [(Frankfurt, London, Sunday, cloudy), 7]. We consider the weights of the target variable as sets of integer value: [(Frankfurt, New York, Sunday, cloudy), $\{1 \ldots 67\}$, [(London, New York, Monday, sunny), $\{68 \ldots 95\}$], [(Frankfurt, London, Sunday, cloudy), $\{96 \ldots 102\}$]. For the feature assignment 'origin'=Frankfurt we want to summarize the set $\{1,\ldots, 67,96, \ldots, 102\}$. (Note that London can be both an origin and destination and we summarize different feature values, i.e., origin_London and destination_London.)

Let V_j denote a disjunction of possible feature value assignments to a given feature f_j. From the summaries we estimate the generalized Jaccard similarity $\frac{\|V_1 \wedge \ldots \wedge V_t\|_\ell}{\|\bigcup(f_i = x_i \in V_1 \vee \ldots \vee V_t)\|_\ell}$ and the union size $\|\bigcup(f_i = x_i \in V_1 \vee \ldots \vee V_t)\|_\ell$ for a profile $P = V_1 \wedge \ldots \wedge V_t$. Each feature value is associated with a set of integers and each V_i with a union of these sets. We use min-wise independent hashing to estimate $\alpha = \frac{|\cap_{i=1}^t A_i|}{|\cup_{i=1}^t A_i|}$ for $t \geq 2$. The sets A_i are the union of several sets $A_i^1, \ldots, A_i^r$: for each set A_i^j we keep a min-wise sample $mws(A_i^j)$ and after processing the stream we take the minimum hash value(s) from $\cup_{j=1}^r mws(A_i^j)$. We will also estimate $|\cup_{i=1}^t A_i|$, thus we can estimate the size of the set intersection $\cap_{i=1}^t A_i$. We refer to the next paragraphs for more details on similarity estimation using minwise independent hashing and set size estimation in data streams. (An overview of state-of-the-art results on set intersection estimation can be found in [12].)

A high level pseudocode description of the data summarization algorithm is in Figure 2. We keep three sketches for each feature assignment $x \in \cup_j X_j$: one for the estimation of $\|P\|_\ell$, $\ell \in \{0, 1, 2\}$. We update the sketches in a streaming fashion. Each new incoming example consisting of k feature assignments updates $3k$ sketches. For each new incoming example $e_i = (\mathbf{x}_i, w(y_i))$ we sample from the set of $w(y_i)^\ell$ and update the sketches for all feature values $x_i \in \mathbf{x}_i$. We estimate the values $\|P\|_\ell$ using the sketches of the features values $x_i \in P$.

Min-wise independent hash functions. The presented algorithms build upon min-wise independent permutations [13]. We give a brief overview of the technique. Assume we are given two sets $A, B \subseteq U$, for a totally ordered universe U. Let $\alpha = \frac{|A \cap B|}{|A \cup B|}$ be the Jaccard similarity between A and B. Define a random permutation $\pi : U \to U$. Let $x = \min(\pi(A \cup B))$, i.e., x is the minimum value under π in $A \cup B$. Let X be an indicator random variable such that $X = 1$ iff $x \in A \cap B$. It is easy to show that the expected value of X is $E[X] = \alpha$. By the sample bound, for $O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$ random permutations and computing the fraction of "minimum" elements from $A \cup B$ that are also in $A \cap B$ yields an (ε, δ)-approximation of α.[1] The algorithm also applies to the generalized Jaccard similarity $\alpha = \frac{|\cap_{i=1}^t A_i|}{|\cup_{i=1}^t A_i|}$ for $t \geq 2$. The approach is applied in a streaming setting by replacing the random permutation π with a suitably defined hash function $h : U \to D$, for some totally ordered set D. A truly random hash function would require to store a random value for each element in the universe U. This leads to considering *approximately min-wise independent hash functions*.

PROCESSSTREAM

Input: stream $\mathcal{S}$ of weighted examples $e_i = ([f_1 = x_1, \ldots, f_k = x_k], w(y_i))$

1: $c_0 = 0, c_1 = 0, c_2 = 0$
2: **for** $e_i \in \mathcal{S}$ **do**
3: **for** $\ell = 0$ to 2 **do**
4: Let R be a min-wise independent sample from $\{c_\ell + 1, \ldots, c_\ell + w(y_i)^\ell\}$.
5: **for** $(f_j = x_j) \in e_i$ **do**
6: Update sketch $sk_\ell[x_j]$ with R.
7: $c_\ell = c_\ell + w(y_i)^\ell$

LEARNTARGETWEIGHT

Input: profile $P = [V_1 \wedge \ldots \wedge V_t]$, power $\ell \in \{0, 1, 2\}$, sketches sk_ℓ

1: J_P = estimate the generalized Jaccard similarity of P from the sketches $sk_\ell[x_1], sk_\ell[x_2], \ldots$ for $x_i \in P$
2: U_P = estimate the size of the union $\|\cup x_i : x_i \in P\|_\ell$
3: **Return** $J_P \cdot U_P$

Fig. 2. The streaming algorithm for estimating $\|P\|_\ell$.

A family $\mathcal{H}$ of functions from a set X to a totally ordered set S, $h : X \to S$, is called *ε-minwise independent* if for any $x \in X$

$$\Pr[h(x) < \min_{y \in X \setminus \{x\}} h(y)] = \frac{1 \pm \varepsilon}{|X|}.$$

For $\varepsilon = 0$ we call h minwise *truly* independent and for $\varepsilon > 0$ minwise *approximately* independent.

We call the above described approach *k-mins sketches* as we store the minimum element for each of k different permutations. A modification of the above approach, the so-called *bottom-k sketches*, stores the k smallest hash values from a given permutation π. Let $\min_k^\pi(A)$ be the k smallest elements in A under π. An estimator of $\alpha = \frac{|\cap_{i=1}^t A_i|}{|\cup_{i=1}^t A_i|}$ is then $\frac{|\cap_{i=1}^t \min_k^\pi(A_i) \cap \min_k^\pi(\cup_{i=1}^t A_i)|}{k}$, i.e., the ratio of the k smallest elements under π which are contained in all sets A_i. It holds $\min_k^\pi(\cup_{i=1}^t A_i) = \min_k(\cup_{i=1}^t \min_k^\pi(A_i))$, thus in a streaming setting we maintain a priority queue for each A_i and store the k smallest elements under π. The advantage of bottom-k

[1] We will present the complexity of the algorithms terms of an unknown parameter α. This is common for data streaming algorithms but might look confusing. Essentially, this can be seen as a short form of the following more precise statement: "Using space $O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$, we can guarantee that (i) if the similarity is at least α, then we obtain an (ε, δ)-approximation, (ii) otherwise, we return a value that is below $(1 + \varepsilon)\alpha$ with probability $1 - \delta$."

sketches is that one needs much less hash functions, and thus faster processing time. However, the hash functions need to be "more random" and more difficult to construct and evaluate, see for example [14].

Note that our specific setting allows to work with minwise truly independent hash functions. We want to evaluate the functions on consecutive unique integers, therefore we can simply generate a random number for each new integer and store the k smallest numbers for a feature assignment. For a sufficiently large number of numbers to draw from, say 2^{64}, this will yield a random permutation of the integers with high probability. The approach works well for estimating the $\|P\|_0$ values but for $\|P\|_1$, and in particular for $\|P\|_2$, it will lead to slow processing time per incoming example. Instead, in Section VII we present efficient algorithms for the evaluation of approximately minwise independent hash function on a sequence of consecutive integers.

Set size estimation in data streams The proposed algorithm uses set size estimation as a subroutine. We apply the approach from [15]. Assume we are given a data stream of integers $u_1, u_2, \ldots$ and we want to estimate the number of different integers. Assume we have a uniform "random enough" hash function $h : \mathbb{N} \to (0, 1]$. Also, we assume that with high probability (whp) the function is injective, i.e., there are no collisions. We will then evaluate $h(u_i)$ for each incoming u_i and store the k smallest hash values. Let v_k be the k-th smallest hash value. An estimate of the number of different integer values is then k/v_k. (If there are less than k different hash values, then we will have the exact value whp.) The intuition is that the more different integers we have the smaller hash values we get. If the hash values are uniformly distributed over the $(0, 1]$ interval, then we expect a fraction of γn to be smaller than $\gamma \in (0, 1]$, n being the number of different values. Thus, we expect $k = v_k n$. It suffices that the function h is only pairwise independent[2] in order to obtain an $(1 \pm \varepsilon)$-approximation with error probability below 1/2 for $k = O(1/\varepsilon^2)$. The median of $O(\log \frac{1}{\delta})$ independent estimates is an (ε, δ)-estimate. We refer to [16] for the optimal bounds on the streaming complexity of the problem.

V. Mean squared error

In this section we show how to approximate the mean and mean-squared error of a given profile P. In the next proofs, we will use the following facts and lemmas:

Fact 1: For any $\varepsilon \in (0, 1/2]$ and constants $c_1, c_2, c_3, c_4 > 0$ there exists $\varepsilon' = O(\varepsilon)$ such that $\frac{(1 \pm c_1\varepsilon')^{c_2}}{(1 \pm c_3\varepsilon')^{c_4}} = (1 \pm \varepsilon)$.

Fact 2: Let $\tilde{q}_1, \tilde{q}_2$ be $1 \pm \varepsilon$-approximations of $q_1, q_2 \geq 1$. Then $\tilde{q}_1 \pm \tilde{q}_2$ is an approximation of $q_1 \pm q_2$ with additive error $\varepsilon(q_1 + q_2)$.

[2]A family $\mathcal{F}$ of functions from U to a finite set S is *k-wise independent* if for a function $f : U \to S$ chosen uniformly at random from $\mathcal{F}$ it holds $\Pr[f(u_1) = c_1 \wedge f(u_2) = c_2 \wedge \cdots \wedge f(u_k) = c_k] = 1/s^k$ for $s = |S|$, distinct $u_i \in U$ and any $c_i \in S$ and $k \in \mathbb{N}$. We call a function chosen uniformly at random from a k-wise independent family *k-wise independent function*.

Lemma 1: Let $\mathcal{H}$ be a family of s ε-minwise independent hash functions $h_i : \mathbb{N} \to (0, 1]$. Let $A_1, \ldots, A_t \subset \mathbb{N}$ be revealed in a streaming fashion in arbitrary order and $\alpha = \frac{|A_i \cap \ldots \cap A_t|}{|A_1 \cup \ldots \cup A_t|}$. Let a_i be the minimum element from $A_i \cup \ldots \cup A_t$ under h_i and q be the number of $a_i \in A_1 \cap \ldots \cap A_t$, $1 \leq i \leq s$. For $s = O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$, $\varepsilon \in (0, 1/2]$, q is an (ε, δ)-approximation of αs.

Proof: Let X_i be an indicator random variable denoting whether $a_i \in A_1 \cup \ldots \cup A_t$, and $X = \frac{1}{s}\sum_{i=1}^{s} X_i$. By definition, $E[X_i] = (1 \pm \varepsilon)\alpha$ and by linearity of expectation $E[X] = (1 \pm \varepsilon)\alpha$. The h_i are independent, thus applying Chernoff's inequality we obtain

$$\Pr[|X - E[X]| \geq \varepsilon E[X]] \leq$$

$$2^{\frac{-\varepsilon^2(1\pm\varepsilon)E[X]s}{2}} = 2^{\frac{-\varepsilon'^2 E[X]s}{2}} = O(\delta)$$

where $\varepsilon' = O(\varepsilon)$. □

Let $sk(A)$ be the k smallest elements in A under a hash function h. Then $\min_k(\cup_i^t sk(A_i))$ is the bottom-sketch of the set $\cup_i^t A_i$. Thus, we keep a bottom-k sketch for each feature assignment $f_i = x$. The following lemma immediately follows from [15].

Lemma 2: Let $\mathcal{H}$ be a family of $s = O(\log \frac{1}{\delta})$ pairwise independent hash function $h_i : \mathbb{N} \to (0, 1]$. Let $A_1, \ldots, A_t \subset \mathbb{N}$ be revealed in a streaming fashion in arbitrary order. Using bottom-sketches of $O(\frac{1}{\varepsilon^2})$ elements for each A_i, we can obtain an (ε, δ)-approximation of $|A_1 \cup \ldots \cup A_t|$.

Theorem 1: Let $\mathcal{S} = e_1, e_2, \ldots$ be a stream of weighted examples. Let P be an arbitrary profile and $\alpha = \operatorname{argmin}_\ell(\frac{\|P\|_\ell}{\|\cup f_i \in P\|_\ell})$. After processing the stream, we can compute an approximation of MSE(P) with additive error $O(\varepsilon\|P\|_2/\|P\|_0)$ and probability $1 - \delta$ using $O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$ space per feature value.

Proof: First observe that the mean weight $\mu(P)$ of the examples complying with a given profile P can be written as $\|P\|_1/\|P\|_0$. We can rewrite

$$MSE(P) = \frac{\sum_{e_i \rhd P}(w(y_i) - \mu(P))^2}{|\{e_i : e_i \rhd P\}|} =$$

$$\frac{\sum_{e_i \rhd P} w(y_i)^2 - 2\mu(P)\sum_{e_i \rhd P} w(y_i) + \mu(P)^2 \sum_{e_i \rhd P} 1}{|\{e_i : e_i \rhd P\}|} =$$

$$\|P\|_2/\|P\|_0 - 2(\|P\|_1/\|P\|_0)^2 + (\|P\|_1/\|P\|_0)^2 =$$

$$\|P\|_2/\|P\|_0 - (\|P\|_1/\|P\|_0)^2.$$

We obtain an (ε, δ)-approximation of $\|P\|_\ell$ by Lemma 1 and Lemma 2. Using Fact 1, we have an (ε, δ)-approximation for the two terms above. By Cauchy-Schwarz inequality we have $\|x\|_2 \geq \sqrt{n}\|x\|_1$ for $x \in \mathbb{R}^n$, thus $\|P\|_0\|P\|_2 \geq \|P\|_1^2$. By Fact 2 the additive approximation is $O(\varepsilon\|P\|_2/\|P\|_0)$. □

VI. Least Absolute Deviation

A drawback of using the mean value for prediction is that it is sensitive to outliers. As an alternative, we can use the median value of examples complying with profile P, $med(P)$. The corresponding error function is the least absolute deviation

$$LAD(P) = \sum_{e_i \rhd P} \frac{|w(y_i) - med(P)|}{\|P\|_0}$$

We refer to [17] for details on learning regression trees using LAD error. In this section we present summarization algorithms for learning regression trees using the LAD error.

Similarly to [18], we consider following definition of median approximation:

Definition. Let U be a totally ordered set and $A \subseteq U$ be a sorted array over n elements. An element $a_i \in A$ is called a positional ε-approximation of median(A) if $(1/2-\varepsilon)n \leq i \leq (1/2+\varepsilon)n$.

We estimate the median using minwise independent hashing based sampling. For each feature value we store the k smallest hash values for a truly independent function, for k to be specified later. Consider a profile $P = (x_1^1 \vee \ldots \vee x_k^1) \wedge \ldots \wedge (x_1^t \vee \ldots \vee x_s^t)$. We compute the k values $w(y_i)$ with the minimum hash values $h(e_i)$ for each feature value x_i. We then find the k values $w(y_i)$ with the minimum hash value of the feature values in each disjunction. We retain the $h(e_i)$ and the corresponding target weights $w(y_i)$ that are presented in all disjunctions in the profile. We return the median of these. Next theorem gives a bound on the required number of samples k.

Theorem 2: Let $A_1, \ldots, A_t$ be weighted sets such that $\alpha = \frac{|A_1 \cap \ldots \cap A_t|}{|A_1 \cup \ldots \cup A_t|}$. Using $k = O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$ minwise independent samples from each A_i we can compute a positional ε-approximation of the median of $A_1 \cap \ldots \cap A_t$.

Proof: Let S_k be the k minwise independent samples from $A_1 \cup \ldots \cup A_t$. The expected number of samples from S_k that also are also in $A_1 \cap \ldots \cap A_t$ is αk. Let $A = A_1 \cap \ldots \cap A_t$ and $n = |A|$. Let $A_< = \{a_i \in A : a_i < \text{med}(A)\}$, $A_\geq = \{a_i \in A : a_i \geq \text{med}(A)\}$. Clearly, $\lfloor n/2 \rfloor \leq |A_<| < |A_\geq| \leq \lceil n/2 \rceil$. The minwise independent hash function is truly random, thus the expected number of samples from $A_<$ and $A_\geq$ is $(\alpha k)/2 \pm 1$. A standard application of Chernoff bounds yields that the median of $k = O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$ samples is a positional ε-approximation of the median with probability $1-\delta$. □

We estimate the LAD error of a given profile P as follows. Let $P_<$ and $P_\geq$ denote the examples $e_i \in P$ such that $w(y_i) < med(P)$ and $w(y_i) \geq med(P)$, respectively. Using that $\|P_<\|_0 \leq |P_\geq\|_0 \leq \|P_<\|_0 + 1$, we obtain

$$LAD(P) = \sum_{e_i \rhd P} \frac{|w(y_i) - med(P)|}{\|P\|_0} =$$

$$\sum_{e_i \rhd P_<} \frac{med(P) - w(y_i)}{\|P\|_0} + \sum_{e_i \rhd P_\geq} \frac{w(y_i) - med(P)}{\|P\|_0} =$$

$$\frac{\|P_\geq\|_1 - \|P_<\|_1}{\|P\|_0} + \frac{med(P)(\|P_<\|_0 - \|P_\geq\|_0)}{\|P\|_0} =$$

$$\frac{\|P_\geq\|_1 - \|P_<\|_1}{\|P\|_0} - \frac{med(P)}{\|P\|_0} = O(\frac{\|P\|_1}{\|P\|_0})$$

Thus, we need to estimate $\|P_<\|_\ell$ and $\|P_\geq\|_\ell$ for $\ell \in \{0,1\}$. Let $\alpha_< = \text{argmin}_\ell(\frac{\|P_<\|_\ell}{\|\cup f_i \in P_<\|_\ell})$, $\alpha_\geq$ is defined in a similar way. Let $\alpha = \min(\alpha_<, \alpha_\geq)$.

Theorem 3: Given a profile P, there exists an algorithm estimating the LAD error for a positional ε-approximation of the median with additive error $O(\varepsilon(med(P) + \|P\|_1/\|P\|_0))$ using space $O(\frac{1}{\alpha\varepsilon^2} \log \frac{1}{\delta})$.

Proof: We will estimate $\alpha_<$ and $\alpha_\geq$ extending the approach from Section V. In addition to the hash values in the minwise sample we also store the value $w(y_i)$, i.e., we store the weight of the example the sample comes from. After processing the stream we first compute the positional approximation of the median. Then we estimate $\alpha_<$ and $\alpha_\geq$ by checking for each sampled hash value whether it is taken from an example of weight less or greater or equal to the estimated median. The claimed bound then follows directly from the proof of Theorem 1 and Theorem 2. □

VII. Processing time.

Of crucial importance for the applicability of the proposed approach is the processing time per incoming example. Given an example e_i with weight $w(y_i)$ and a hash function $h : \mathbb{N} \to (0,1]$, we need to find the minimum hash value of evaluating the function h on $w(y_i)^\ell$ consecutive integers. For larger values of $w(y_i)$ and $\ell \in \{1,2\}$ this might be prohibitively expensive. Existing results on weighted sampling with applications to weighted Jaccard similarity estimation [19], [20] have limitations and are not applicable to our setting. The algorithm from [19] applies to min-k sketches such that we can generate a sample from a weighted set in constant time. However, it does not extend to bottom-k sketches. Haeupler et al. [20] present an algorithm that applies to Jaccard similarity estimation using bootom-k sketches. However, the algorithm assumes that the size of the weighted sets are known in advance and does not apply to a real-time streaming setting.

Rigorous theoretical results for min-k sketches can be obtained when implementing the hash functions h_j using *tabulation hashing* [21]. Assume all keys come from a universe $\mathcal{U}$ of size n. With tabulation hashing, we view each key $r \in \mathcal{U}$ as a vector consisting of c characters, $r = (r_1, r_2, \ldots, r_c)$, where the i-th character is from a universe $\mathcal{U}_i$ of size $n^{1/c}$. (W.l.o.g. we assume that $n^{1/c}$ is an integer). For each universe $\mathcal{U}_i$, we initialize a table T_i and for each character $r_i \in \mathcal{U}_i$ we store a random value v_{r_i}. Then the hash value is computed as:

$$h_0(r) = T_1[r_1] \oplus T_2[r_2] \oplus \ldots \oplus T_c[r_c]$$

where $\oplus$ denotes the bit-wise XOR operation. Thus, for a small constant c, the space needed is $O(n^{1/c} \log n)$ bits and the evaluation time is $O(1)$ array accesses. For example, keys are 64-bit integers and $c = 4$. Tabulation hashing yields only 3-wise independent hash functions. However, as shown by [22], it yields ε-minwise independent hash functions with $\varepsilon = O(\frac{\log^2 n}{n^{1/c}})$.

In order to design algorithms for the fast update of k-mins and bottom-k sketches we consider the following problems.

Definition 1: MinHashValue(W, q, κ): Given a hash function $h : \mathbb{N} \to (0, 1]$ and $\kappa, q \in \mathbb{N}, q \leq W$, find the minimum value in $\{h(\kappa + 1), \ldots, h(\kappa + q)\}$, $q \leq W$.

Definition 2: MinKHashValues(W, q, κ, k, τ): Given a hash function $h : \mathbb{N} \to (0, 1]$ and $\kappa, q \in \mathbb{N}, q \leq W$, find the minimum at most k values in $\{h(\kappa+1), \ldots, h(\kappa+q)\}$, $q \leq W$, which are smaller than τ.

The following theorem shows that we can efficiently solve the above problems.

Theorem 4: Let $h : \mathbb{N} \to (0, 1]$ be implemented using tabulation hashing with parameters n and c. Let $W \leq n^{1/c}$. After preprocessing in time $O(W \log W)$ and space $O(W)$ we can solve the MinHashValue(W, q, κ) in time $O(\log W)$. The MinHashValue(W, q, κ, τ) can be solved in time $O(k+\log W)$ after preprocessing in time and space $O(W \log W)$.

Proof: Since $W \leq n^{1/c}$, we can assume that for $\{h(\kappa + 1), \ldots, h(\kappa + q)\}$ there are at most two different possibilities of the tables $T_1, \ldots, T_{c-1}$, i.e., the leading bits of the integer numbers $\kappa + 1, \ldots, \kappa + q$ might change only once. Thus, we need a data structure that will support queries like "Given a bit vector b, find the element x in D such that $b \oplus x$ is minimal and $rank(x) \leq rank(b)$." In a preprocessing phase we build a binary search tree B consisting of value-rank pairs (v, r) supporting queries of the form "Given a query (q, r_q), output the pair $(v, r) = \text{argmin}_v(v \geq q, r \leq r_q)$. There are W pairs such that $(v, r) = (h(i), i)$, $1 \leq i \leq W$. Pairs are compared according to the value v. The root of each subtree records the minimum rank of a pair in the subtree. We perform standard search for the smallest $v \geq q$ and at each internal node we check the rank of the subtree that contains the elements. If the minimum rank is more than the query rank r_q, then all elements in the subtree are outside the query range. In such a case either inspect the other branch or backtrack. We can only backtrack if the tree contains both elements smaller and larger than q. In this case, we will reach a subtree with elements larger than q and minimum rank larger that r_q. However, this is a unique tree and we might backtrack at most once. Once we have reached a tree where all elements are larger than q and the minimum rank is less than r_q, we find in time $O(\log W)$ the smallest element with rank less than r_q. The tree B can be build in time $O(W \log W)$ and needs space $O(W)$.

For MinKHashValues(W, q, κ, k, τ) we present a data structure that supports range queries of the form "Given a query (q, r_q), output the (at most) k smallest values in the interval $[q, q+\tau]$ which have rank at most r_q. We build again a binary search tree B. At each root we additionally store an array consisting of the elements in the subtree sorted according to their rank. We then determine the intervals, i.e., the nodes in B, which cover $[q, q+\tau]$. There are at most $\log W$ such intervals which can be found in time $O(\log W)$. Let I be the list of found intervals. The intervals in I are pairwise disjoint and all elements in a given interval are strictly smaller or larger than all elements in the other intervals. We output the elements from I starting with the leftmost interval until the rank of the output element is less than r_q. We stop when we either have output k elements or inspected all intervals in I. The time for sorting the elements and the space usage is $O(W \log W)$ and once we have identified the set of relevant intervals, each hash value can be computed in constant time. □

VIII. Discretizing numeric features.

For real-valued features, instead of selecting a subset of the possible feature assignments, we have to select a value s and split the data depending on whether the given feature value is smaller or larger than s. We can discretize real values such that we preserve as much as possible the quality of the original splits. Assume feature values are drawn from a universe U of cardinality u. We consider different discretization options.

Fixed summarization points. Feature value v is projected to v div k. This results in u/k feature values. Another option is that a feature value v is projected to $\hat{v} = \lfloor \log_{1+\gamma} v \rfloor$ for a user-defined $\gamma > 0$. This assures that $(1 + \gamma)^{\hat{v}}$ will be an $(1 - \gamma)$-approximation of v. The total number of values is bounded by $\log_{1+\gamma} u = O(\frac{1}{\gamma} \log u)$. These methods are static in the sense that the discretization is independent from the data distribution. However, for certain types of numeric features they might yield good results, in particular if we have some a priori information on the values.

Mergeable histograms. A more advanced method is to adapt the algorithm from [3]. We maintain a histogram that dynamically adapts to the distribution of the feature values. The histogram consists of b bins, for a user-defined b. For a new feature value a new bin is created. If the number of bins exceeds b, then the two bins closest to each other are merged. As discussed, the minwise independent sketches can be merged, thus the algorithm applies to our setting. The algorithm is heuristical and no precise bounds on the quality of the approximation can be obtained. However, it is empirically shown to yield excellent results for a variety of distributions.

Approximate quantiles. Given an ordered set D of n elements, $d \in D$ is an *ε-approximate ϕ-quantile* if it has a rank between $(\phi - \varepsilon)n$ and $(\phi + \varepsilon)n$. We build upon the q-digest algorithm [23]. Assume values are in the range $[1, \sigma]$. We maintain a data structure that represents a binary tree with σ leafs. An inner node t corresponds to a given interval $[t.min, t.max]$. Each node y has a counter $t.count$ for the number of values in $[t.min, t.max]$. A new value is assigned to a leaf and the counter is updated. Let n be the number of values seen so far in the stream and $t.l$ and $t.r$ be the left and right child of an internal node t. For each such t we maintain the invariant that $t.count \leq \varepsilon n$ and $t.count+t_l.count+t_r.count > \varepsilon n$. If the condition is violated, then we merge the three nodes t, t_l, t_r into t and add up the counters. In this way, we explicitly store at most $1/\varepsilon$ leafs. The intuition is that non-frequent values will be collected in higher-level nodes, as these contribute less to correctly identifying approximate quantiles. Looking for a ϕ-quantile, we traverse the tree in post-order by increasing $t.max$ values. Once for

some t the sum of the counts reaches ϕn, we return $t.max$ as an ε-approximate ϕ-quantile.

From the q-digest data structure we obtain a list of $1/\phi$ ε-approximate ϕ-quantiles $\phi n, 2\phi n, \ldots$. In addition to the counts, at each node we store the min-wise samples. When we merge the nodes, we also update the minwise samples in the same way as when computing the minwise sample of a CNF. We thus can estimate the error when splitting on approximate ϕ-quantiles for arbitrary data distribution.

IX. Experiments

We implemented the algorithm in Python and performed experiments on a Linux machine with an Intel 2.6 GHz clocked CPU with 4MB of CPU cache and 8 GB of main memory. The hash functions were implemented using tabulation hashing for a universe of size 2^{64} and $c = 4$. The 4 tables consist each of 2^{16} random numbers and can be loaded in fast CPU cache. The random numbers are from the Marsaglia Random Number CDROM[3]. We used bottom-k sketches with the efficient hash function evaluation presented in Section VII.

Purpose of the experiments. The main goal of the experimental evaluation is to provide evidence that the presented sketching algorithms can be of practical importance. We show that using bottom-k sketches we obtain a good approximation of the desired quantities. We would like to note that rigorous theoretical results hold only for min-k sketches as tabulation hashing is only 3-wise independent [22]. However, min-k sketches lead to very slow processing time. It is well-known that on real data hash functions often work better than suggested by the conservative theoretical analysis and explaining this behavior is an active research area, c.f. [24], [25]. We also present a comparison to sampling that will demonstrate the advantages and limitations of our approach.

Datasets. We used three datasets for the experimental evaluation: Flights, Network and Synthetic. The Flights dataset[4] consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. We selected three features, Origin, Destination and Carrier. The Network dataset is an internal dataset that describes network packets described by different discrete features and a measure of interest, similarly to the setting in [6]. The Synthetic dataset is an artificial dataset we created as follows. We generated examples consisting of three features. Each feature value configuration has a corresponding target weight sampled from a normal distribution $\mathcal{N}(\mu, \mu/2)$ where $\mu \in [100, 200]$ is a random number. Triples of feature values are randomly generated by drawing features from a uniform distribution. In addition, a small number of randomly selected feature value triples are generated such that their Jaccard similarity and is considerably higher than for triples with independently generated features. More precisely, for feature values x_1, x_2, x_3 and an example e_i, $\Pr[e_i = (x_1, x_2, x_3)] \gg \Pr[x_1 \in e_i] \cdot \Pr[x_2 \in e_i] \cdot \Pr[x_3 \in e_i]$, i.e., we create statistically significant feature configurations. The datasets are summarized in Table I; μ and σ refer to the mean and standard deviation of the example weights.

[3]http://stat.fsu.edu/pub/diehard/cdrom/

[4]http://stat-computing.org/dataexpo/2009/

Dataset	# examples	# features	μ	med	σ	LAD
Flights	$\approx 5.8 \cdot 10^7$	696	23.94	12	37.02	18.6
Network	$\approx 1.2 \cdot 10^7$	27	821.57	340	6852	278
Synthetic	10^7	30	157.1	143	129	145

TABLE I
Information on evaluation datasets. μ and σ denote the mean and standard deviation of the target weights.

Results. We give an overview of the achieved results in terms of scalability and quality of approximation.

Running time. We compared the running time for the implementation that explicitly evaluates the hash function $w(y_i)^\ell$ times to our improved implementation. For the Flights dataset the first 3 million examples are processed in about 160 minutes and 5 minutes, respectively, including the preprocessing time. The time savings for Synthetic are somewhat smaller, 30 vs. 5 minutes. For the Network dataset, we needed one hour to process less than 15,000 examples when applying the explicit hash function evaluation. Note that in order to estimate $\|P\|_2$, we considered only the 8 most significant bits of the example weights. This form of discretization increases the approximation error but, as evident from the evaluation, we still obtain very good results.

Approximation guarantees. In Figures 3 we plot the approximation of $\|P\|_1$ and $\|P\|_2$ of profile P with three feature values disjunctions from Flights data for the 10 runs of the algorithm. The sketch size is 5,000. It is visible that we obtain an unbiased estimator. In Figure 4 we plot the approximation of the mean for two different profiles $P_{(2)}$ and $P_{(3)}$ where $P_{(2)}$ consists of two feature value disjunctions and $P_{(3)}$ – of three feature value disjunctions from the Network dataset. As evident from the plots, we obtain an unbiased estimator for both profiles and, as expected, the variance of the estimation depends on the Jaccard similarity of the profiles.

We summarize the experimental results in Table II , Table III for MSE and Table IV for LAD. Let $\tilde{q}$ be an estimate of q. For a quantity q and its estimate $\tilde{q}$, we define the approximation error as $|q - \tilde{q}|/q$.

Given a profile P, we report the approximation error for $\|P\|_\ell$, $\mu(P)$ and $MSE(P)$. The approximation is the median of the approximations from 10 individual runs of the algorithm for 4 different sketch sizes. In Table II we chose at random a profile with a conjunctions of two feature dishunctions and in Table III – a profile with three disjunctions. We see the algorithm achieves very good approximation for the $\|P\|_\ell$ values and the mean. However, the additive factor $\|P\|_2/\|P\|_0$ can be larger, and thus even if for excellent estimation of the $\|P\|_\ell$ values the MSE estimation is sometimes not so precise. In Table IV we summarize the estimation we obtain for the median and the LAD error for the same profiles as in Table III. As we see, the median estimates are very accurate and thus we obtain a good estimation for LAD.

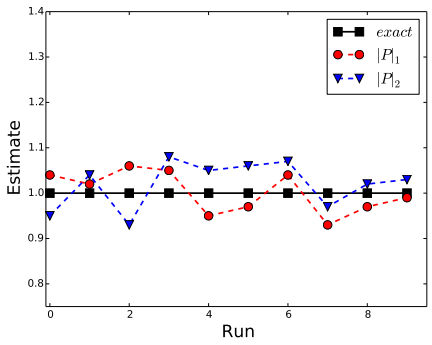

Fig. 3. Estimates for $\|P\|_1$ and $\|P\|_2$.

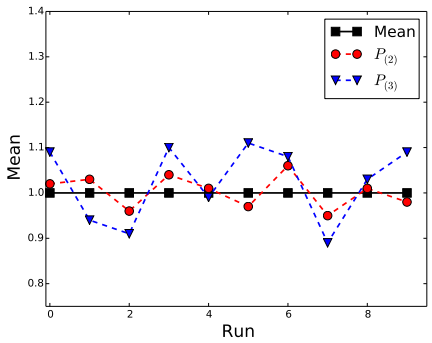

Fig. 4. Estimates of the mean for profiles $P_{(2)}$ and $P_{(3)}$ with Jaccard similarity 0.182 and 0.066, respectively.

Dataset	s	$\|P\|_0$	$\|P\|_1$	$\|P\|_2$	μ	MSE
Flights	1,000	0.031	0.045	0.023	0.014	0.065
	2,500	0.006	0.033	0.003	0.039	0.107
	5,000	0.013	0.015	0.004	0.028	0.057
	10,000	0.001	0.022	0.006	0.021	0.019
Network	1,000	0.009	0.012	0.006	0.004	0.011
	2,500	0.002	0.026	0.007	0.025	0.049
	5,000	0.016	0.017	0.007	0.002	0.019
	10,000	0.007	0.010	0.002	0.003	0.025
Synthetic	1,000	0.013	0.006	0.026	0.007	0.044
	2,500	0.005	0.013	0.018	0.018	0.061
	5,000	0.007	0.006	0.011	0.014	0.046
	10,000	0.006	0.016	0.002	0.022	0.066

TABLE II
SUMMARY OF RESULTS FOR PROFILES CONSISTING OF 2 FEATURE DISJUNCTIONS. THE JACCARD SIMILARITY FOR FILGHTS IS 0.103, FOR NETWORK – 0.297, AND FOR SYNTHETIC – 0.142.

Dataset	s	$\|P\|_0$	$\|P\|_1$	$\|P\|_2$	μ	MSE
Flights	1,000	0.111	0.007	0.089	0.094	0.215
	2,500	0.012	0.019	0.033	0.009	0.052
	5,000	0.023	0.012	0.006	0.025	0.028
	10,000	0.006	0.022	0.009	0.017	0.018
Network	1,000	0.021	0.028	0.021	0.048	0.172
	2,500	0.028	0.008	0.023	0.035	0.032
	5,000	0.025	0.012	0.018	0.037	0.072
	10,000	0.014	0.019	0.06	0.005	0.138
Synthetic	1,000	0.038	0.018	0.074	0.029	0.112
	2,500	0.028	0.006	0.036	0.037	0.105
	5,000	0.018	0.012	0.032	0.033	0.078
	10,000	0.004	0.007	0.005	0.016	0.074

TABLE III
SUMMARY OF RESULTS FOR PROFILES CONSISTING OF 3 FEATURE DISJUNCTIONS. THE JACCARD SIMILARITY FOR FILGHTS IS 0.019, FOR NETWORK – 0.066, AND FOR SYNTHETIC – 0.085.

Space savings. For the given datasets we compute the compression ratio as the number of elements in the sketches divided by the number of elements in the original data. Thus, for FLIGHTS we get for sketch size 1,000 and running 10 copies of the algorithm in parallel a compression ratio of about 4%, and for sketch size 10,000 the compression ratio is about 40%. For the other two datasets the compression is better due to the smaller number of features: between 0.75% and 7.5% for NETWORK and between 1% and 10% for SYNTHETIC.

Comparison to sampling. In Figure 5 we plot the estimates for the mean for different profiles with three feature values disjunctions we obtain from standard sampling and our approach. We use a summary of size 1,000 for each feature and a sample of the data corresponding to roughly the same size. As we see, for Flights we obtain considerably better estimates from sampling. On the other hand sketching yields better estimates for the Synthetic and Network datasets. The reason is as follows. Let p be the fraction of examples complying with the given profile and α the Jaccard similarity of the profile. Let N be the number of examples in the sample and M the number of values in each sketch, respectively. Clearly, $M < N$. The distribution of weights for Flights profile is stable and in a random sample we expect more occurrences of the profile, i.e., $pN > \alpha M$. The profile from the synthetic dataset appears in less than 1% of the examples, but the Jaccard similarity is about 25%. For the network profile there exist a few examples with much larger than the majority of the examples. Therefore, if some of these examples are not sampled, we are likely to underestimate the mean. In our algorithm we consider all weights and thus obtain unbiased estimates.

X. CONCLUSIONS AND FUTURE DIRECTIONS

Leveraging and extending upon state-of-the-art data summarization techniques, we presented new algorithms for regression tree learning in data streams. We experimentally confirmed the theoretical findings. A future research direction is a thorough experimental evaluation on datasets with different characteristics. Moreover, an evaluation of the different discretization approaches presented in VIII is necessary in order to fully understand the capabilities and limitations of the proposed algorithms. We learned a regression tree from the sketches as described in [11]. The results look promising but do not add much more insight than the results presented in the paper. In order to develop a scalable system that can handle massive data streams at scale, one would ideally design intelligent algorithms that can decide in real time whether sketching or sampling is a better alternative for the problem at hand.

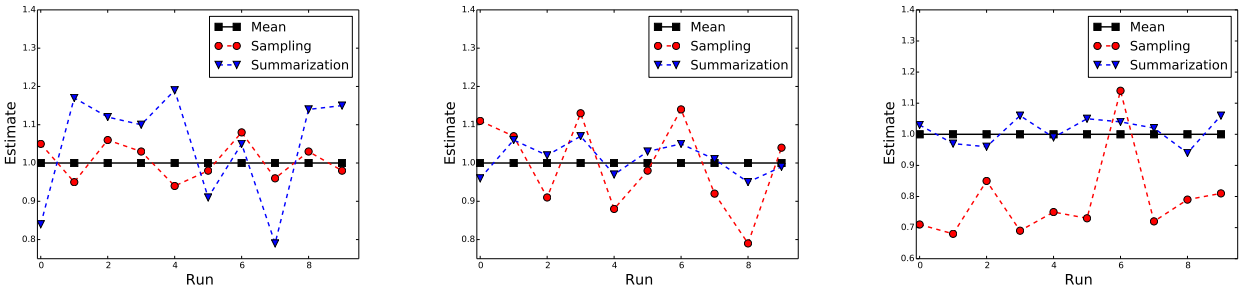

Fig. 5. Estimates of the mean obtained by sampling and sketching for Flights(left), Synthetic(middle) and Network(right).

Dataset	s	$\|P_<\|_0$	$\|P_>\|_0$	$\|P_<\|_1$	$\|P_>\|_1$	med	LAD
Flights	1,000	0.132	0.092	0.011	0.089	0.05	0.121
	2,500	0.012	0.082	0.016	0.051	0.00	0.0867
	5,000	0.014	0.036	0.029	0.02	0.00	0.0657
	10,000	0.008	0.017	0.009	0.013	0.00	0.0547
Network	1,000	0.101	0.098	0.089	0.041	0.019	0.198
	2,500	0.081	0.085	0.052	0.053	0.013	0.112
	5,000	0.045	0.042	0.033	0.044	0.009	0.075
	10,000	0.018	0.021	0.018	0.027	0.00	0.048
Synthetic	1,000	0.038	0.067	0.028	0.074	0.106	0.172
	2,500	0.028	0.012	0.021	0.031	0.068	0.098
	5,000	0.019	0.008	0.013	0.017	0.043	0.055
	10,000	0.016	0.008	0.009	0.002	0.045	0.039

TABLE IV
SUMMARY OF RESULTS FOR PROFILES CONSISTING OF 3 FEATURE DISJUNCTIONS. THE JACCARD SIMILARITY FOR FILGHTS IS 0.019, FOR NETWORK – 0.066, AND FOR SYNTHETIC – 0.085.

REFERENCES

[1] P. M. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

[2] E. Ikonomovska, J. Gama, B. Zenko, and S. Dzeroski, "Speeding-up hoeffding-based regression trees with options," in *Proceedings of the 28th International Conference on Machine Learning, ICML*, 2011, pp. 537–544.

[3] Y. Ben-Haim and E. Tom-Tov, "A streaming parallel decision tree algorithm," *J. Mach. Learn. Res.*, vol. 11, pp. 849–872, 2010.

[4] S. Kpotufe and F. Orabona, "Regression-tree tuning in a streaming setting," in *27th Annual Conference on Neural Information Processing Systems, NIPS.*, 2013, pp. 1788–1796.

[5] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, 1999.

[6] Z. Yu, Z. Ge, A. Lall, J. Wang, J. J. Xu, and H. Yan, "Crossroads: A practical data sketching solution for mining intersection of streams," in *Proceedings of the 2014 Internet Measurement Conference, IMC*, 2014, pp. 223–234.

[7] D. Feldman, M. Schmidt, and C. Sohler, "Turning big data into tiny data: Constant-size coresets for *k*-means, PCA and projective clustering," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2013, pp. 1434–1453.

[8] E. Liberty, "Simple and deterministic matrix sketching," in *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013*, 2013, pp. 581–588.

[9] N. Pham and R. Pagh, "Fast and scalable polynomial kernels via explicit feature maps," in *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013*, 2013, pp. 239–247.

[10] A. Shrivastava and P. Li, "Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS)," in *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI*, 2015, pp. 812–821.

[11] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[12] R. Pagh, M. Stöckel, and D. P. Woodruff, "Is min-wise hashing optimal for summarizing set intersection?" in *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14*, 2014, pp. 109–120.

[13] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.

[14] G. Feigenblat, E. Porat, and A. Shiftan, "Exponential time improvement for min-wise based algorithms," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2011, pp. 57–66.

[15] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Randomization and Approximation Techniques, 6th International Workshop, RANDOM*, 2002, pp. 1–10.

[16] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, 2010, pp. 41–52.

[17] L. F. R. A. Torgo, "Inductive learning of tree-based regression models," Ph.D. dissertation, 1999.

[18] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, USA.*, 1998, pp. 426–435.

[19] S. Ioffe, "Improved consistent sampling, weighted minhash and L1 sketching," in *ICDM 2010, The 10th IEEE International Conference on Data Mining*, 2010, pp. 246–255.

[20] B. Haeupler, M. Manasse, and K. Talwar, "Consistent weighted sampling made fast, small, and easy," *CoRR*, vol. abs/1410.4266, 2014.

[21] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143 – 154, 1979.

[22] M. Pătraşcu and M. Thorup, "The power of simple tabulation hashing," *J. ACM*, vol. 59, no. 3, p. 14, 2012.

[23] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004*, 2004, pp. 239–249.

[24] K. Chung, M. Mitzenmacher, and S. P. Vadhan, "Why simple hash functions work: Exploiting the entropy in a data stream," *Theory of Computing*, vol. 9, pp. 897–945, 2013.

[25] M. Thorup, "Bottom-k and priority sampling, set similarity and subset sums with minimal independence," in *Symposium on Theory of Computing Conference, STOC'13*, 2013, pp. 371–380.